Computing architectures

M. Haefele, M. Lobet

Maison de la Simulation

Acknoledgments: G. Hager, G. Wellein, M. Klemm

Observatoire de Paris, October 8 2018

Why this course ?

Why are you sitting here ?

Purpose of this course

- **Why is it important to measure performance ?**

⇒ During **code optimization** process, ”measuring is better than guessing”, Brian Wylie, developer of Scalasca

- **Why is it important to have an optimized code ?**

⇒ To get results faster

⇒ To ensure the best utilization of High Performance Computing (HPC) infrastructures

⇒ To get access to HPC infrastructures

Outline

- HPC architectures & performance bottlenecks
- Performance evaluation concepts & methodology
- Presentation of the code under study
- Scalasca
- Optimisation at the core level
- VTune / Advisor

Debunking ideas

Lore 1

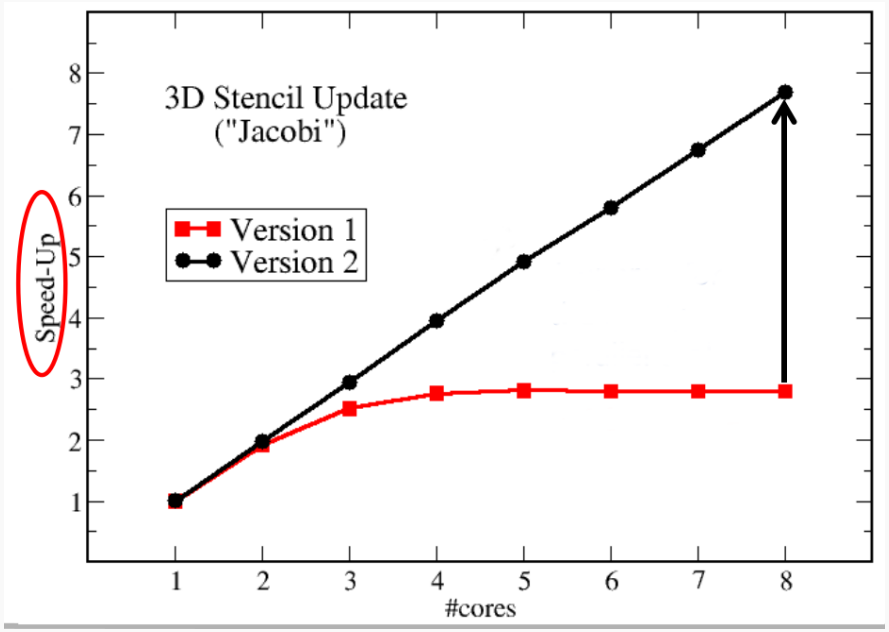
In a world of highly parallel computer architectures only highly scalable codes will survive

Lore 2

Single core performance no longer matters since we have so many of them and use scalable codes

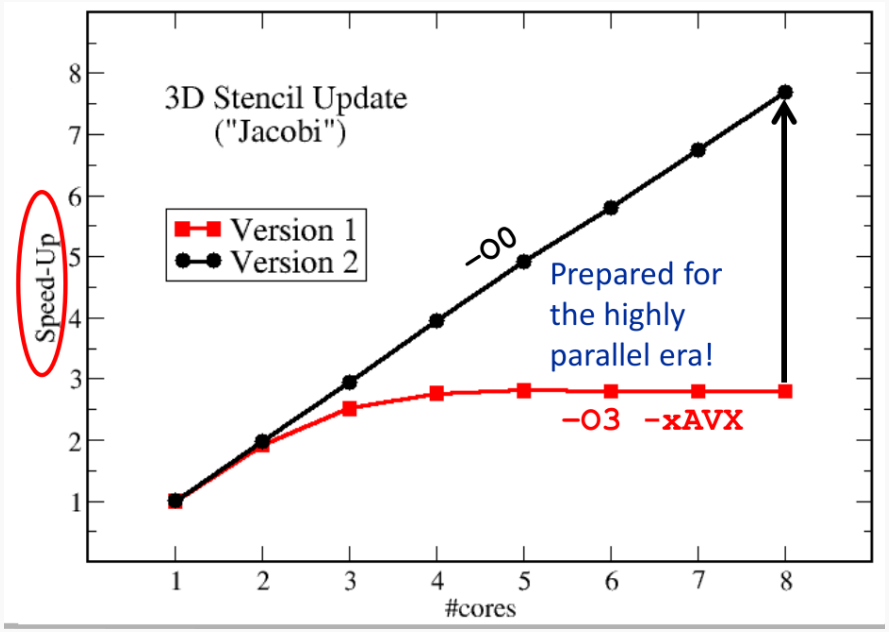
Debunking ideas

```
!$OMP PARALLEL DO
do k = 1 , Nk
   do j = 1 , Nj; do i = 1 , Ni
      y(i,j,k)= b*( x(i-1,j,k)+ x(i+1,j,k)+ x(i,j-1,k)+
                    x(i,j+1,k)+ x(i,j,k-1)+ x(i,j,k+1))
   enddo; enddo
enddo
```



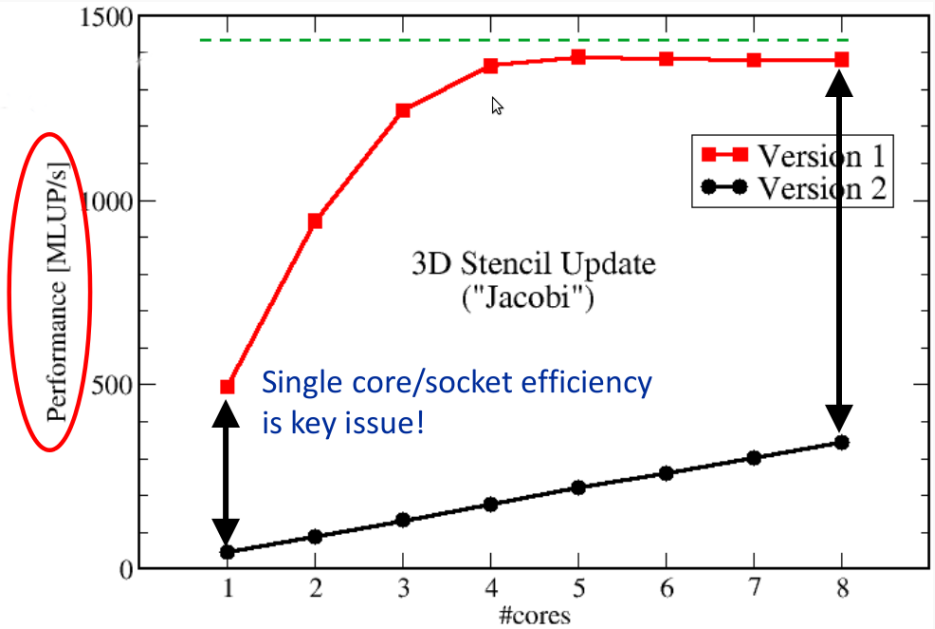
Debunking ideas

```
!$OMP PARALLEL DO
do k = 1 , Nk
  do j = 1 , Nj; do i = 1 , Ni
    y(i,j,k)= b*( x(i-1,j,k)+ x(i+1,j,k)+ x(i,j-1,k)+
                  x(i,j+1,k)+ x(i,j,k-1)+ x(i,j,k+1))
  enddo; enddo
enddo
```



Debunking ideas

```
!$OMP PARALLEL DO
do k = 1 , Nk
   do j = 1 , Nj; do i = 1 , Ni
      y(i,j,k)= b*( x(i-1,j,k)+ x(i+1,j,k)+ x(i,j-1,k)+
                    x(i,j+1,k)+ x(i,j,k-1)+ x(i,j,k+1))
   enddo; enddo
enddo
```



Debunking ideas

HPC **is not only** about scalability !

HPC **is about** running at the **bottleneck of the hardware** !

Bottlenecks

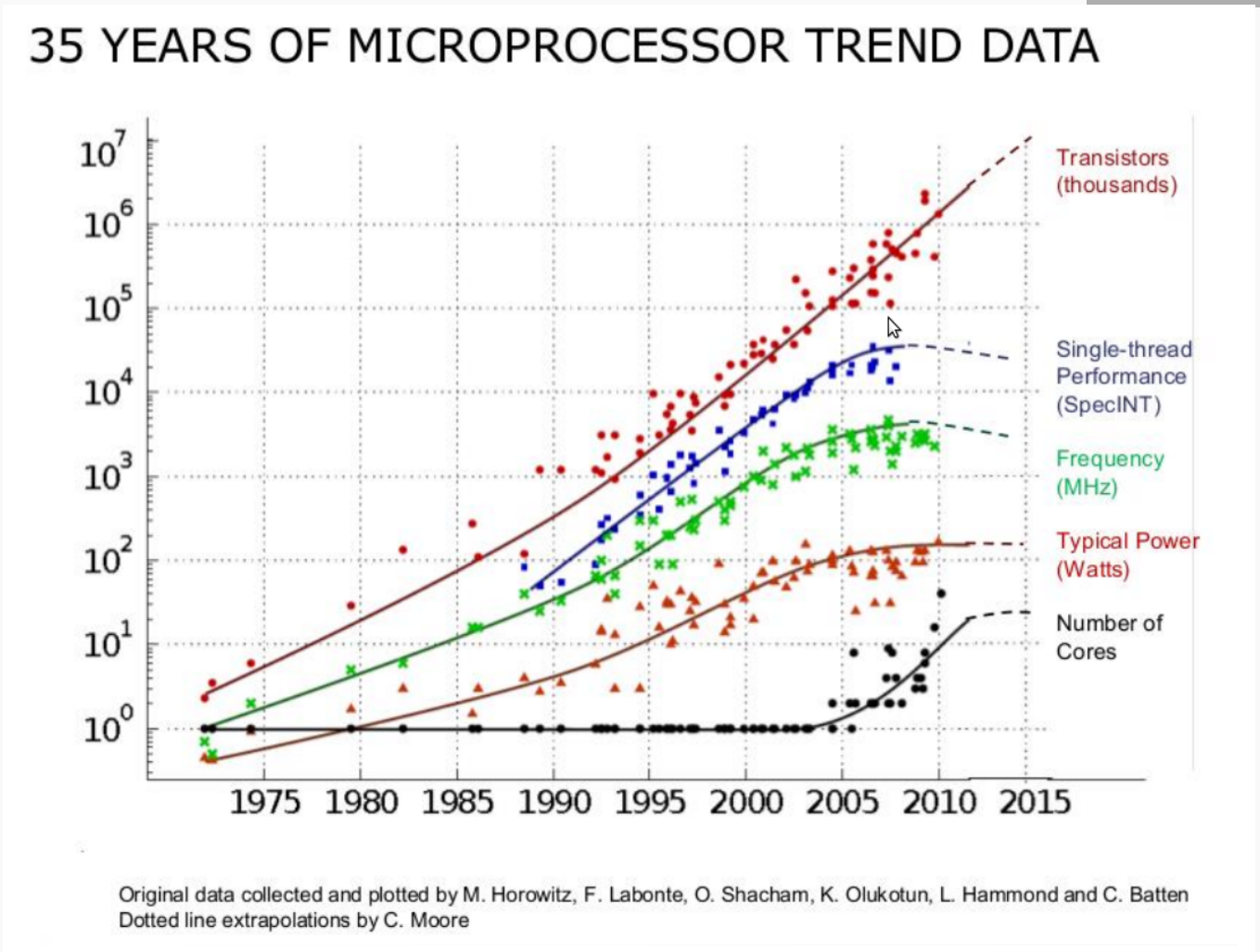
Hierarchical studies for hierarchical architectures

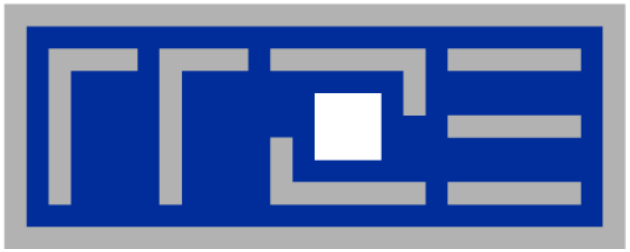
- Core: Computing unit
- Node: Shared memory unit
- Communications: Distributed memory environment
- Input/Output: File system access

Outline

- Recent CPU architectures
  - General architecture of a cached based processor
  - Pipeline
  - Superscalar processors (ILP)
  - Simultaneous multi-threading (SMT)
  - Single Instruction Multiple Data (SIMD)
  - Memory hierarchy
  - UMA vs ccNUMA
  - Peak performance
- Intel Xeon Phi KNL (RIP)
- IBM OpenPower: IBM Power CPU + NVidia GPU
- Comparison

Moore's law and parallelism





Node-Level Performance Engineering

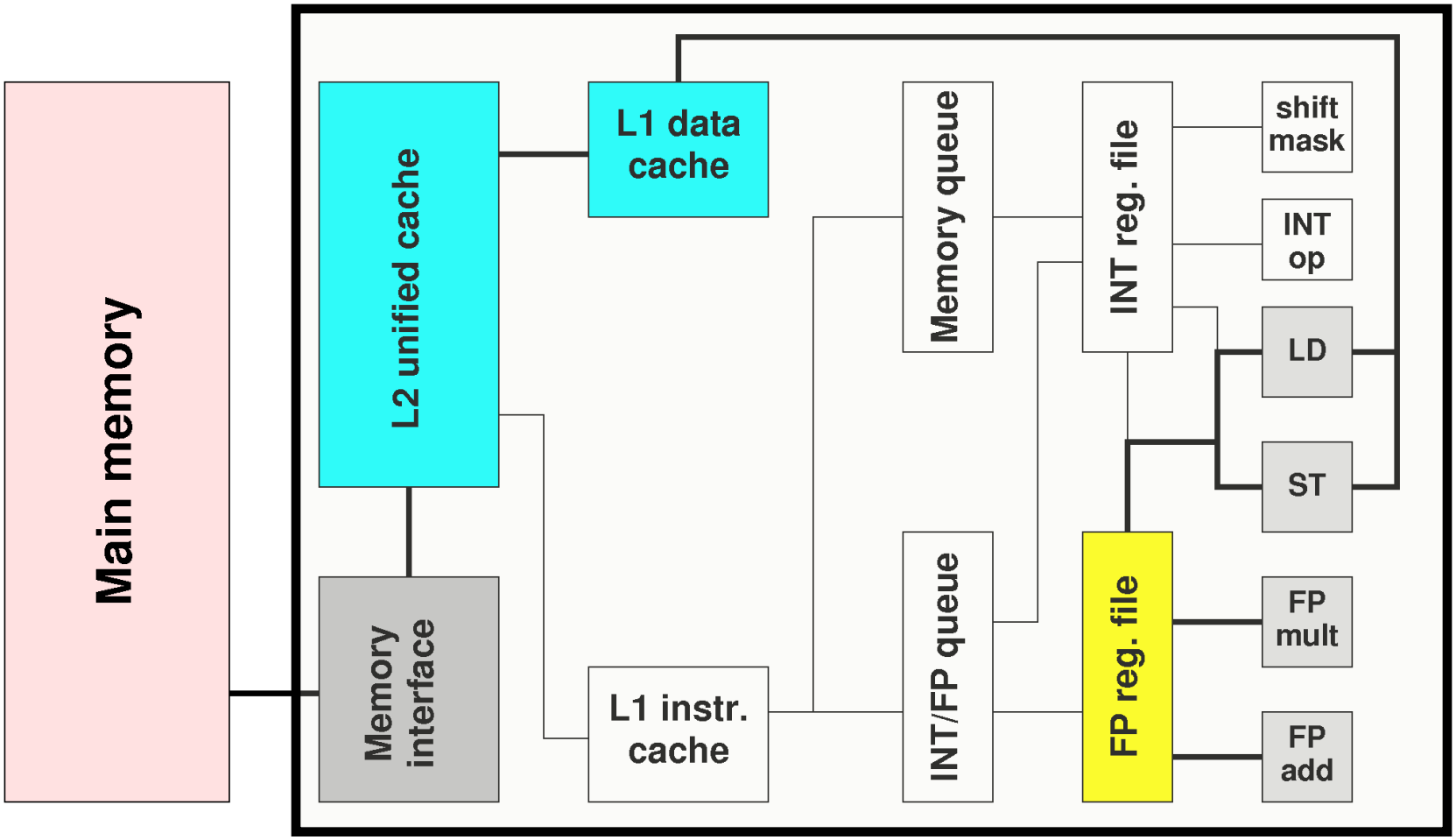
http://goo.gl/4kS16

Georg Hager, Gerhard Wellein
Erlangen Regional Computing Center
University of Erlangen-Nuremberg

Two-day short course
LRZ Garching
3./4.12.2013

General-purpose cache based microprocessor core

- **(Almost) the same basic design in all modern systems**



Not shown: most of the control unit, e.g. instruction fetch/decode, branch prediction,…

Pipelining of arithmetic/functional units

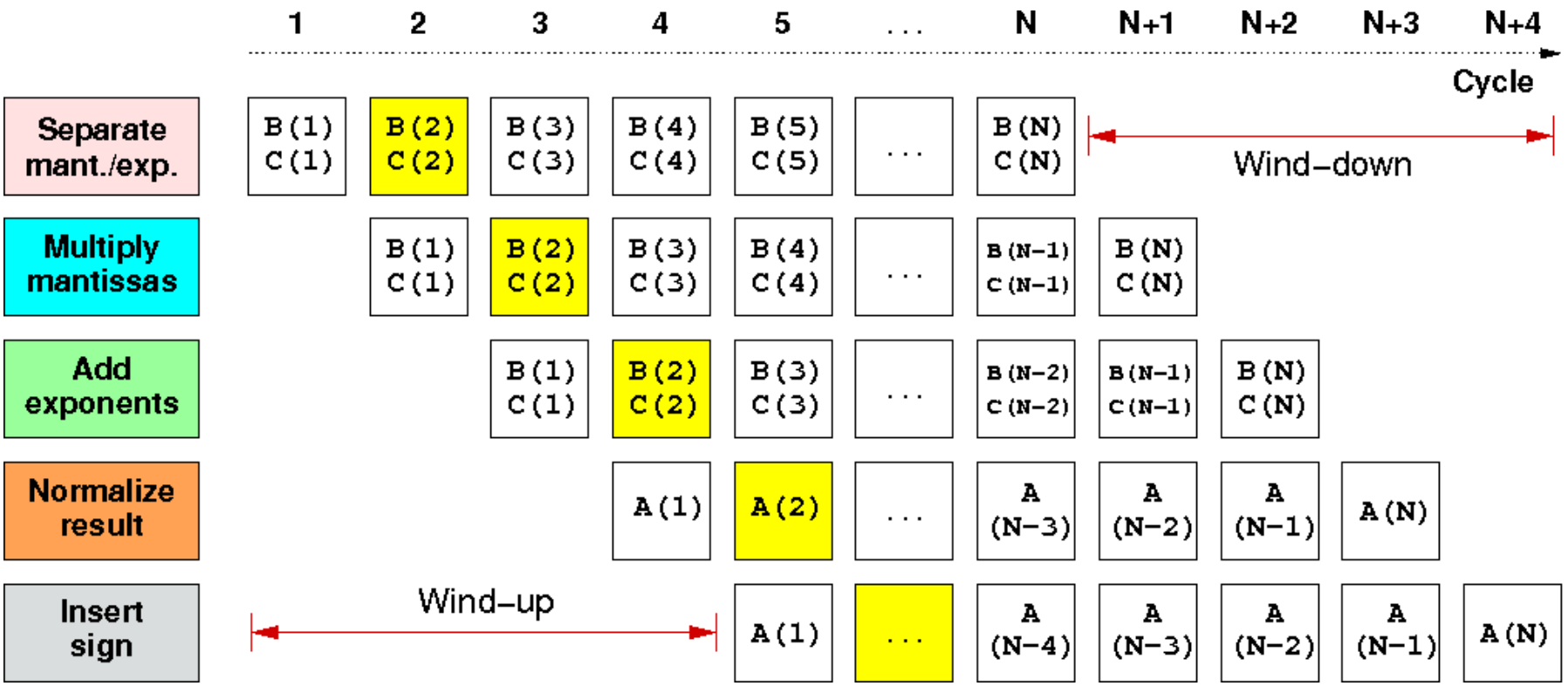
- **Idea:**
  - Split complex instruction into several simple / fast steps (stages)
  - Each step takes the same amount of time, e.g. a single cycle
  - Execute different steps on different instructions at the same time (in parallel)

- **Allows for shorter cycle times (simpler logic circuits), e.g.:**
  - floating point multiplication takes 5 cycles, but
  - processor can work on 5 different multiplications simultaneously
  - one result at each cycle after the pipeline is full

- **Drawback:**
  - Pipeline must be filled - startup times (#Instructions >> pipeline steps)
  - Efficient use of pipelines requires large number of independent instructions → instruction level parallelism
  - Requires complex instruction scheduling by compiler/hardware – software-pipelining / out-of-order

- **Pipelining is widely used in modern computer architectures**

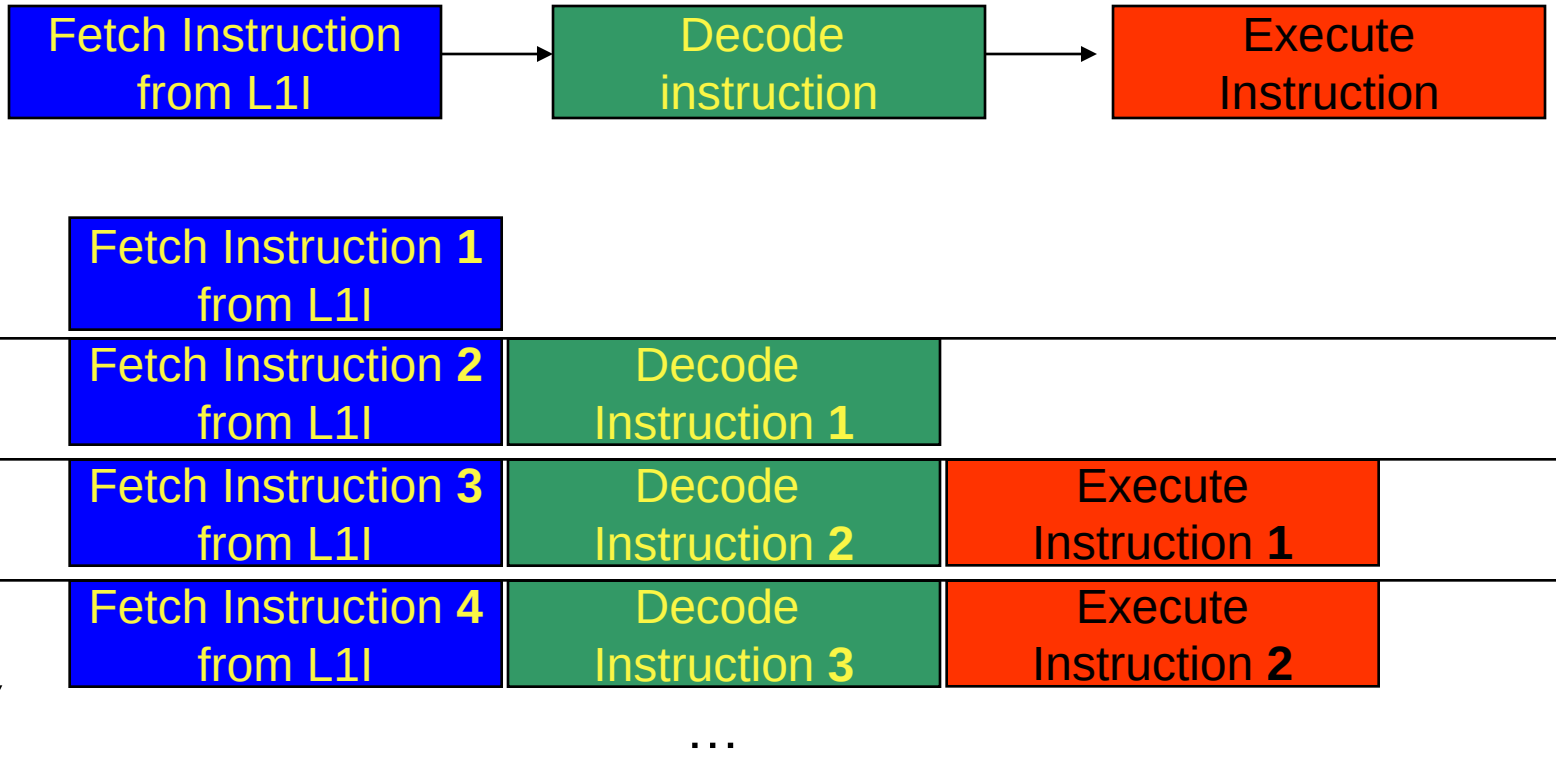
5-stage Multiplication-Pipeline: A(i)=B(i)*C(i) ; i=1,...,N



First result is available after 5 cycles (=latency of pipeline)!

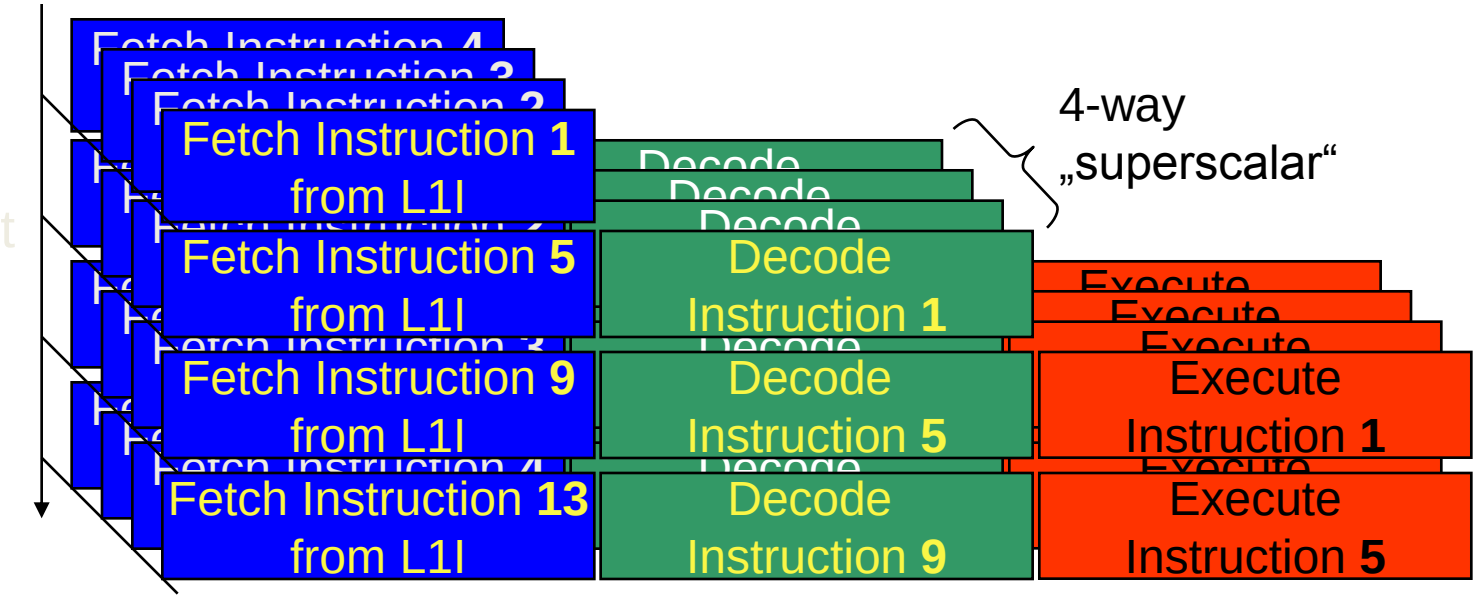
Wind-up/-down phases: Empty pipeline stages

- **Besides arithmetic & functional unit, instruction execution itself is pipelined also, e.g.: one instruction performs at least 3 steps:**



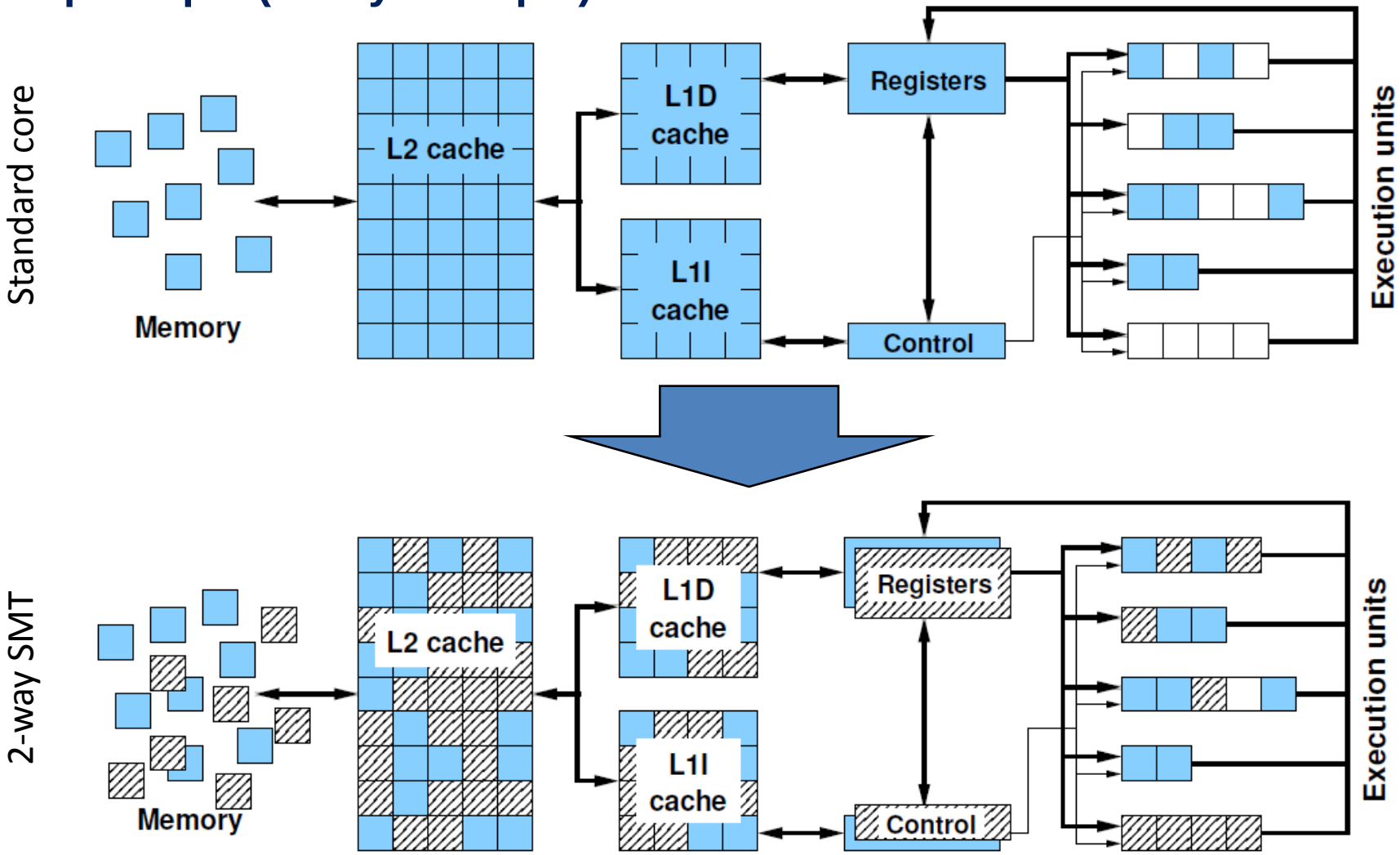
- Branches can stall this pipeline! (Speculative Execution, Predication)
- Each unit is pipelined itself (e.g., Execute = Multiply Pipeline)

- Multiple units enable use of Instrucion Level Parallelism (ILP): Instruction stream is “parallelized” on the fly

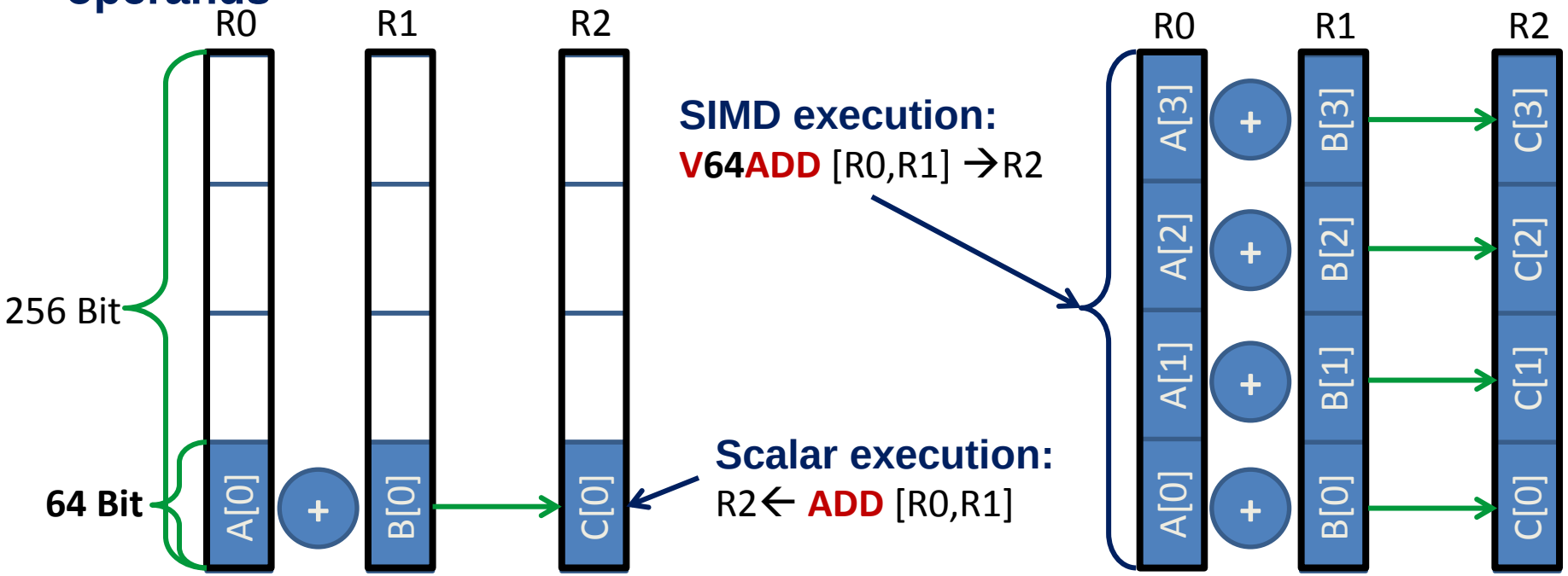


- Issuing m concurrent instructions per cycle: m-way superscalar
- Modern processors are 3- to 6-way superscalar & can perform 2 floating point instructions per cycles

SMT principle (2-way example):



- **Single Instruction Multiple Data (SIMD) operations allow the concurrent execution of the same operation on "wide" registers**
- **x86 SIMD instruction sets:**
  - SSE: register width = 128 Bit → 2 double precision floating point operands
  - AVX: register width = 256 Bit → 4 double precision floating point operands
- **Adding two registers holding double precision floating point operands**

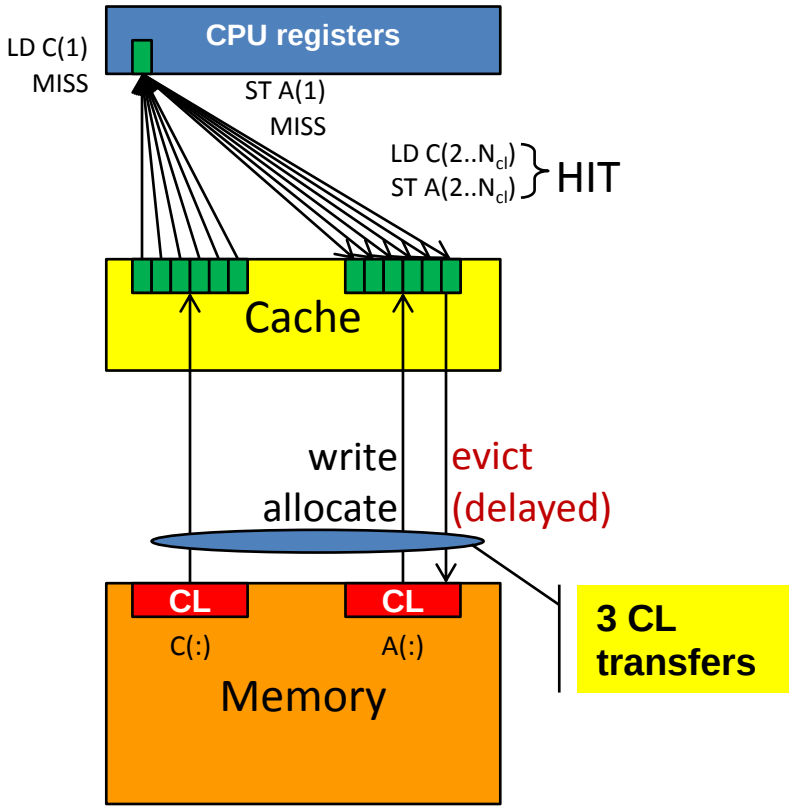


Registers and caches: Data transfers in a memory hierarchy

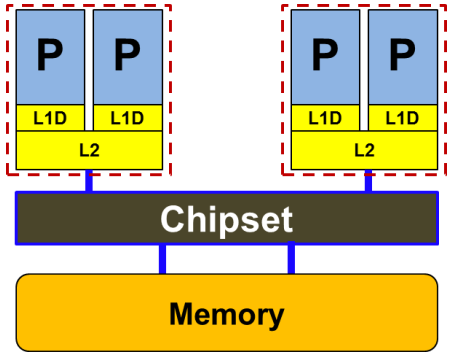
- How does data travel from memory to the CPU and back?

- Remember: Caches are organized in cache lines (e.g., 64 bytes)
- Only complete cache lines are transferred between memory hierarchy levels (except registers)
- MISS: Load or store instruction does not find the data in a cache level
  → CL transfer required

- Example: Array copy `A(:)=C(:)`



Yesterday (2006): Dual-socket Intel "Core2" node:

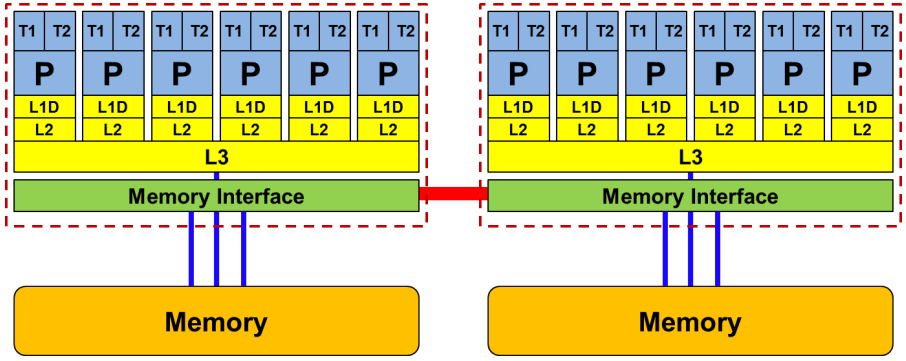


Uniform Memory Architecture (UMA)

Flat memory ; symmetric MPs

But: system "anisotropy"

Today: Dual-socket Intel (Westmere,…) node:

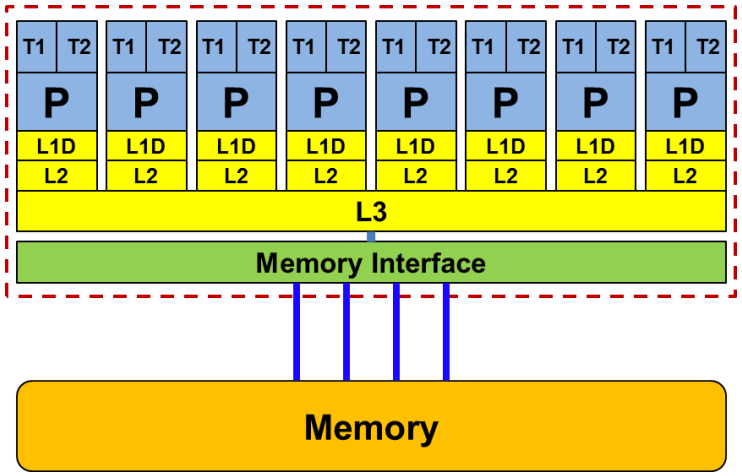


Cache-coherent Non-Uniform Memory Architecture (ccNUMA)

HT / QPI provide scalable bandwidth at the price of ccNUMA architectures: *Where does my data finally end up?*

On AMD it is even more complicated → ccNUMA within a socket!

There is no single driving force for chip performance!



Intel Xeon
"Sandy Bridge EP" socket
4,6,8 core variants available

Floating Point (FP) Performance:

$$P = n_{core} * F * S * \nu$$

n_{core}	number of cores:	8
F	FP instructions per cycle: (1 MULT and 1 ADD)	2
S	FP ops / instruction: (256 Bit SIMD registers – "AVX")	4 (dp) / 8 (sp)
ν	Clock speed :	~2.7 GHz

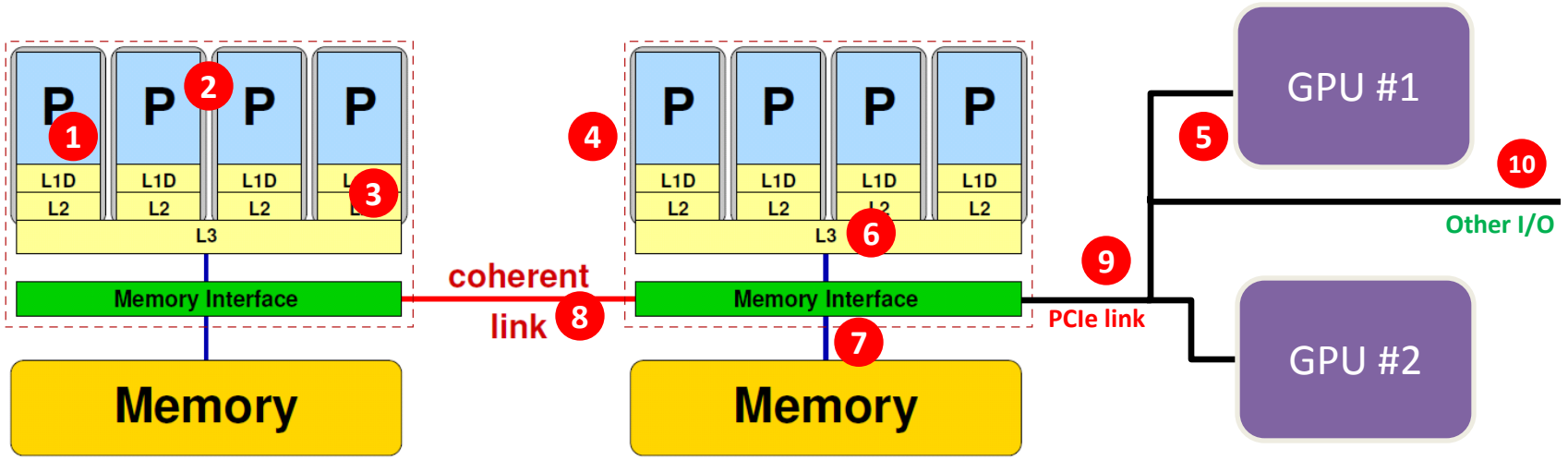
TOP500 rank 1 (mid-90s)

P = 173 GF/s (dp) / 346 GF/s (sp)

But: P=5.4 GF/s for serial, non-SIMD code

Parallelism in a modern compute node

- **Parallel and shared resources within a shared-memory node**



Parallel resources:

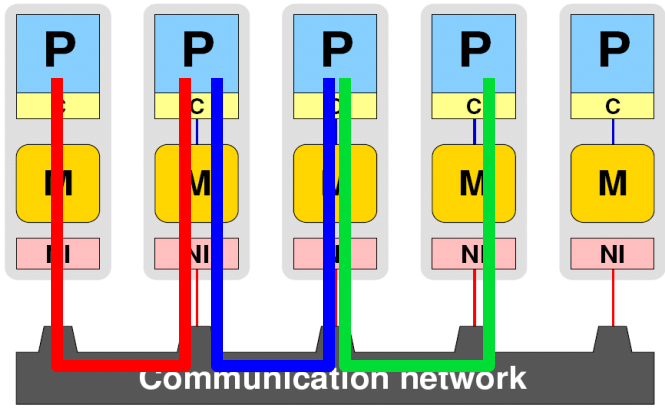
- Execution/SIMD units 1
- Cores 2
- Inner cache levels 3
- Sockets / ccNUMA domains 4
- Multiple accelerators 5

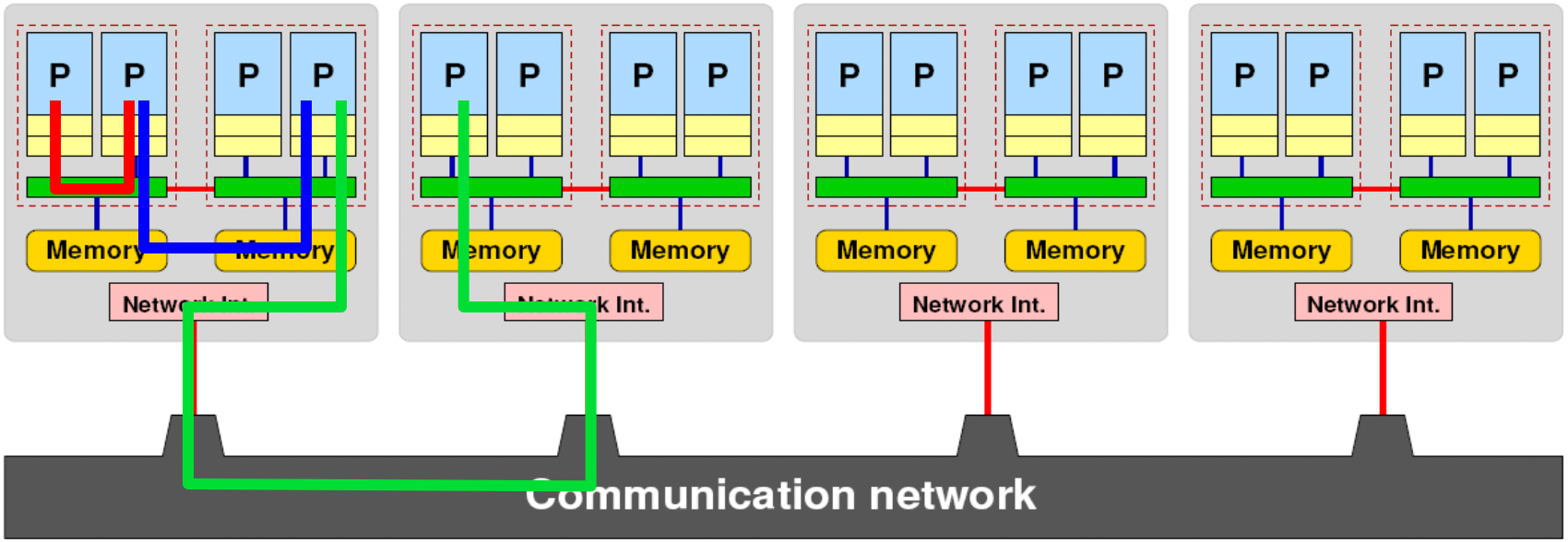
Shared resources:

- Outer cache level per socket 6
- Memory bus per socket 7
- Intersocket link 8
- PCIe bus(es) 9
- Other I/O resources 10

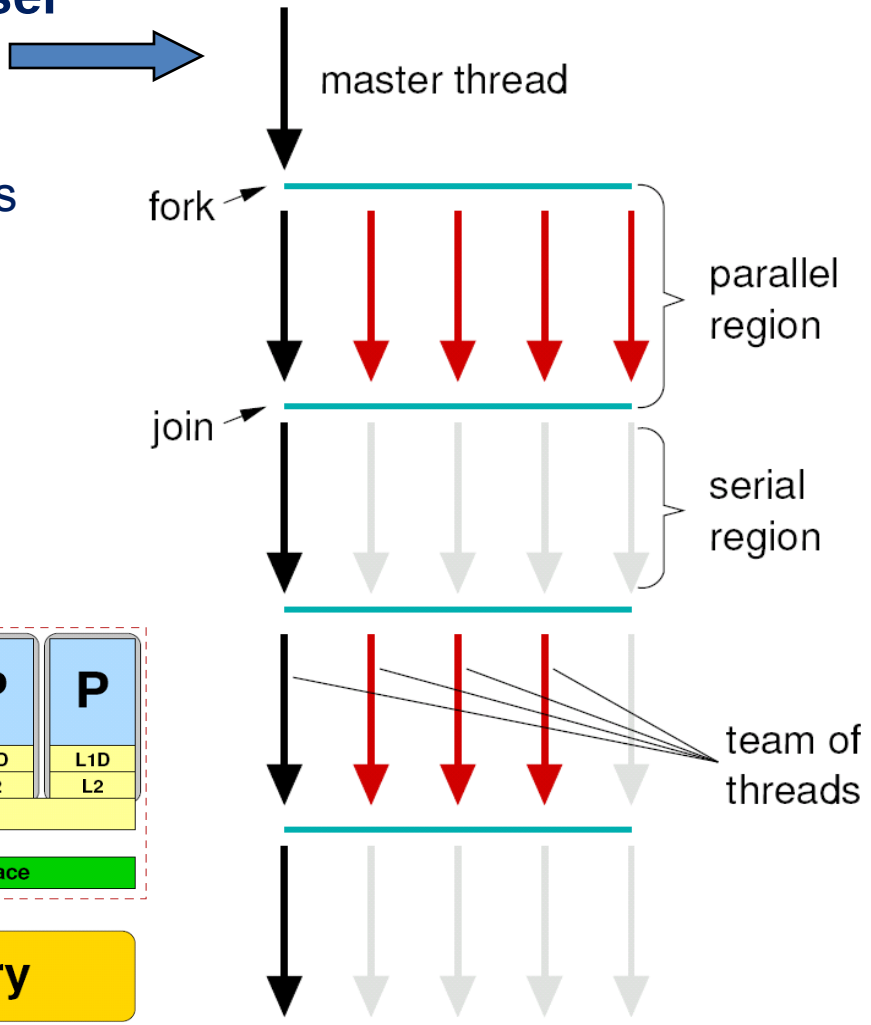
How does your application react to all of those details?

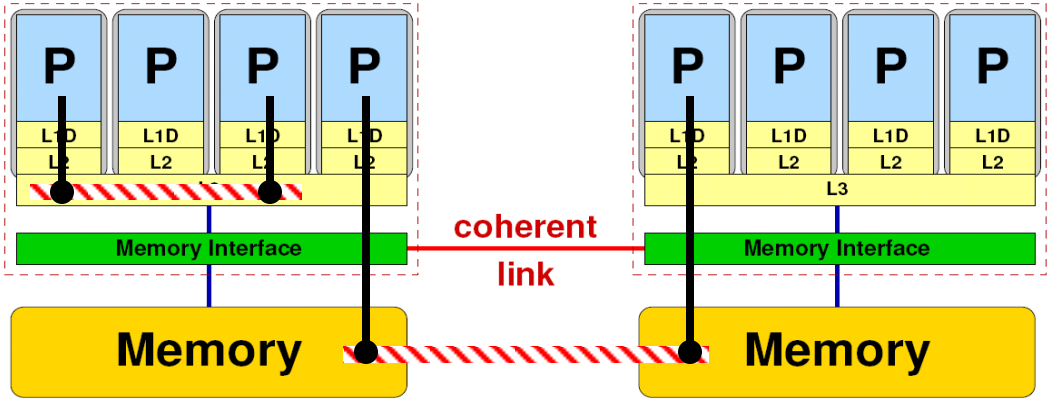
- **Machine structure is invisible to user:**
  - → Very simple programming model
  - → MPI "knows what to do"!?
- **Performance issues**
  - Intranode vs. internode MPI
  - Node/system topology





- **Machine structure is invisible to user**
  - → Very simple programming model
  - Threading SW (OpenMP, pthreads, TBB,…) should know about the details
- **Performance issues**
  - Synchronization overhead
  - Memory access
  - Node topology





Parallel programming models: Lots of choices

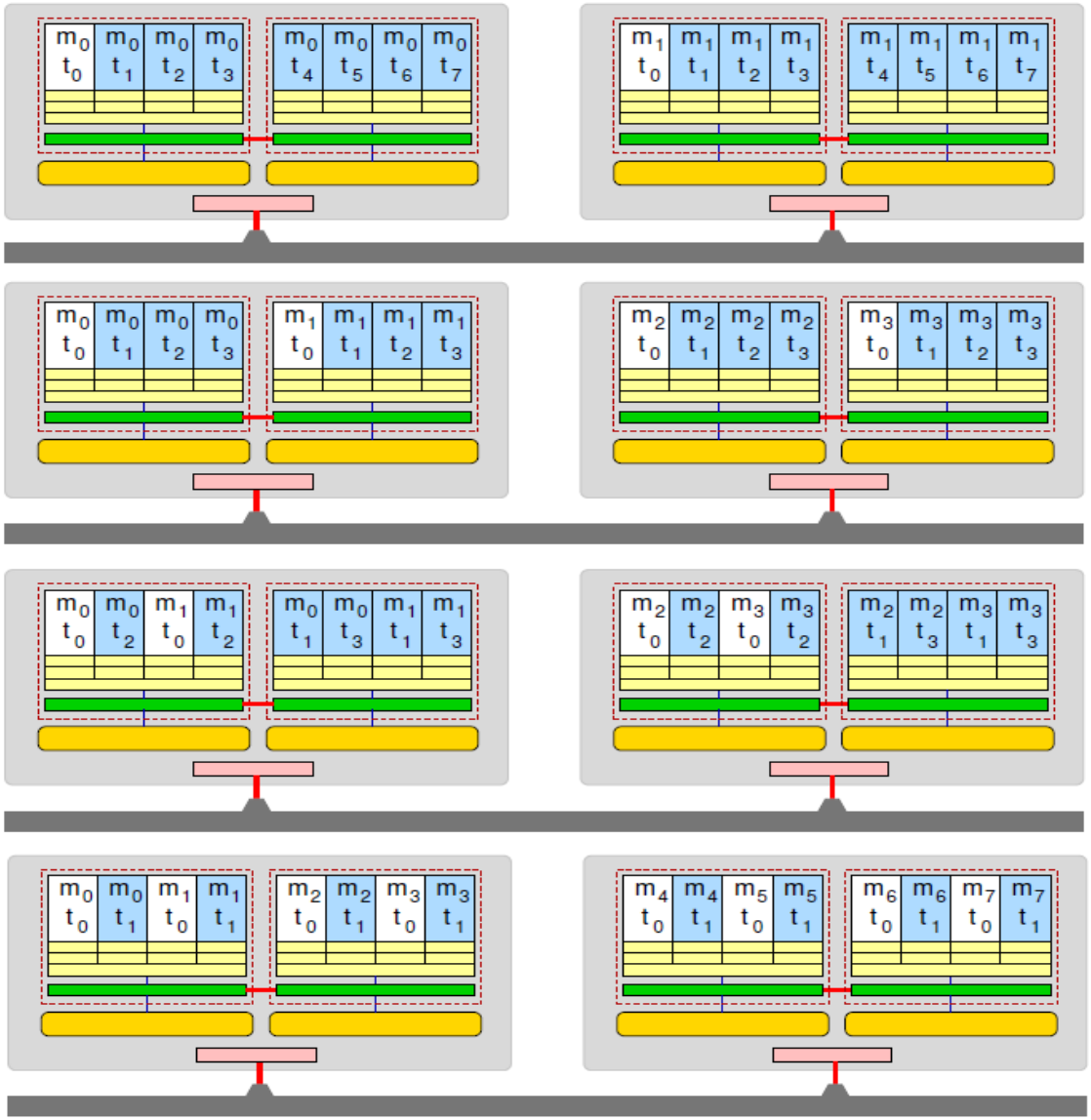
Hybrid MPI+OpenMP on a multicore multisocket cluster

One MPI process / node

One MPI process / socket:
OpenMP threads on same socket: **“blockwise”**

OpenMP threads pinned **“round robin”** across cores in node

Two MPI processes / socket
OpenMP threads on same socket



File systems

- Shared resources $\Rightarrow$ your job can be affected by another job running next to it
- Different file systems on a single HPC platform
  - Home: back-uped, low performance
  - Work: back-uped or not, medium-high performance
  - Scratch: no back-up, high performance
- ASCII vs binary output
- Serial vs parallel IO
- File system bandwidth reachable vs total bandwidth

KNL Architecture Overview

ISA

Intel® Xeon® Processor Binary-Compatible (w/Broadwell)

On-package memory

Up to 16GB, ~500 GB/s STREAM at launch

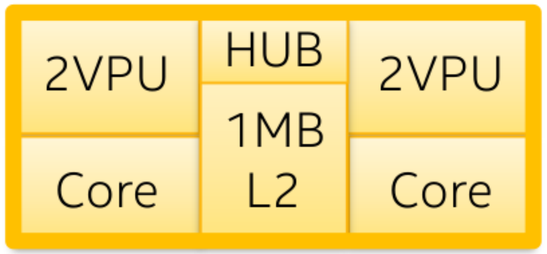
Platform Memory

Up to 384GB (6ch DDR4-2400 MHz)

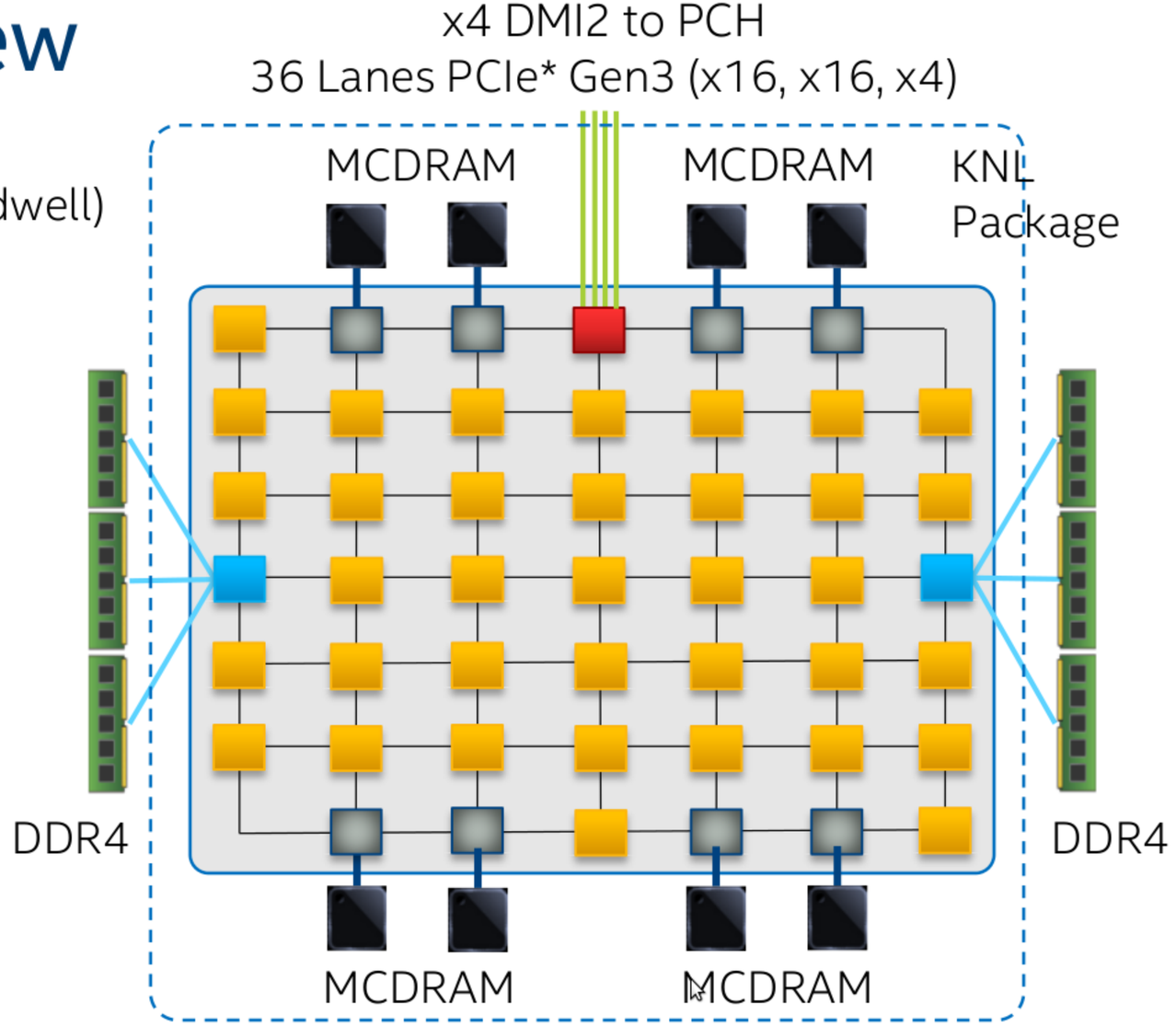
Fixed Bottlenecks

- ✓ 2D Mesh Architecture
- ✓ Out-of-Order Cores
- ✓ 3x single-thread vs. KNC

TILE: (up to 36)



Enhanced Intel® Atom™ cores based on Silvermont™ Microarchitecture



Tile | EDC (embedded DRAM controller) | IMC (integrated memory controller) | IIO (integrated I/O controller)

NVLINK TO CPU

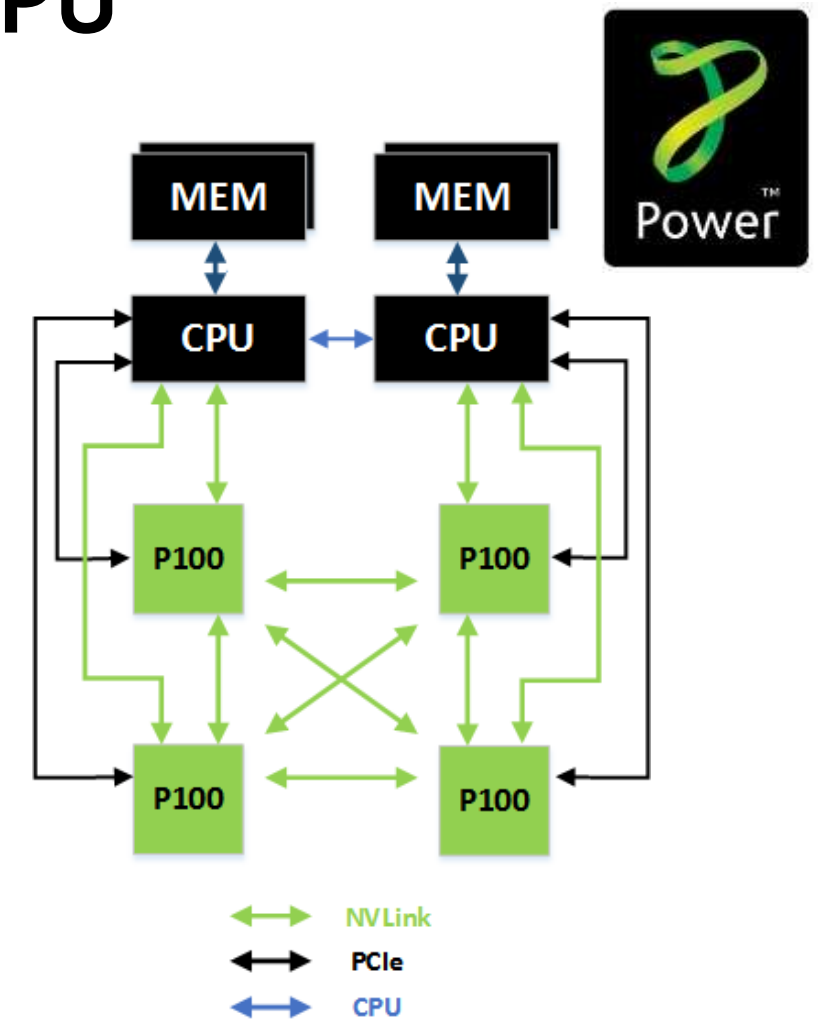
Fully connected quad

120 GB/s per GPU bidirectional for peer traffic

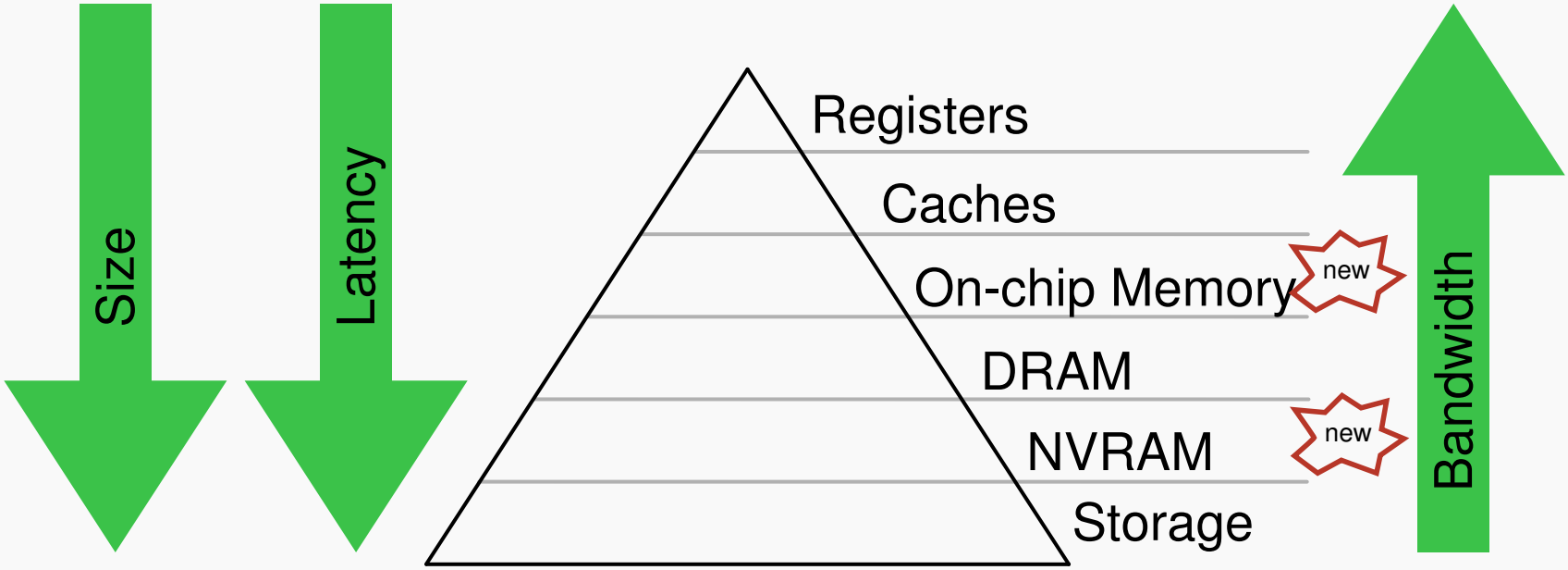
40 GB/s per GPU bidirectional to CPU

Direct Load/store access to CPU Memory

High Speed Copy Engines for bulk data movement



Memory hierarchy



Architecture comparison

Device	2x Skylake 8168	Intel KNL 7250	NVidia P100	Xilinx XCVU9P
Techno.	14nm	14nm	16nm	16nm
Freq.	2.7 GHz	1.4 GHz	1.5 GHz	0.1-0.5 GHz
Power	410W	215W	300W	$<$ **50W**
#cores	48	68	3584?	N.A
Cache	57 MiB	34 MiB	18 MiB	62 MiB
Fast mem	N.A.	16 GB	16 GB HBM	8? GB HBM
Mem	128-512 GB	384 GB	N.A.	48GB
Peak perf. (DB)	2 TF/s	2 TF/s	5.3 TF/s	$>$ **0.5TF/s**

FPGA: similar performance for 10x less energy !